LTAT.05.003
Software Engineering

Lecture 04: Analysis

Dietmar Pfahl

email: dietmar.pfahl@ut.ee

Fall 2020
Lectures 5 + 6: Flipped Classroom

- Lecture 5 (02-Oct-2020) - Development Infrastructure - Development/Build/Collaboration Tools
- Lecture 6 (09-Oct-2020) - Continuous Development & Integration

We will use ‘flipped classroom’ for these two lectures:
- You watch the video recordings (from last year) posted on the course wiki BEFORE the lecture
- If you have specific questions (e.g., requesting clarification about the materials presented in the recorded lecture) you post them at the latest on the day before the lecture time in Slack (general channel)
- Kristiina Rahkema will be available at the beginning of the normal lecture time (10:15 on 02 Oct and 09 Oct) to answer questions posted in Slack and new questions arising during the lecture. Once there are no more questions, the lecture is over.

Kristiina Rahkema
Labs Next Week

- Assessment of your Homework 2 progress
- Each project team must attend with all its members present
- If not all team members present, penalty applies

Via Zoom!
Labs this week …

Lab 2 Assignment:

1. Project Planning
   - Refinement of requirements into Tasks
   - Prioritization of Tasks (P1, P2, P3, P4)
   - Responsibility assignment
   - Effort estimation

2. Five Use Case Descriptions

3. (Initial) Domain Model
Schedule of Lectures

Week 01: Introduction to SE
Week 02: Requirements Engineering I
Week 03: Requirements Engineering II
**Week 04: Analysis**
Week 05: Development Infrastructure
Week 06: Continuous Development and Integration
Week 07: Project Estimation / Architecture and Design I
Week 08: Architecture and Design II
Week 09: Verification and Validation I
Week 10: Verification and Validation II
Week 11: Refactoring (and TDD)
Week 12: Agile/Lean Methods
Week 13: Industry Guest Lecture
Week 14: Course wrap-up, review and exam preparation
Week 15: Reserve time slot (no lecture scheduled as of today)
Acknowledgement

Textbooks/Slides:

- Ivan Marsic: Software Engineering, 2012
  (http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf)

Who has looked at the text book, e.g., wrt. Use Case Descriptions?
Structure of Lecture 04

• Preliminaries and Context
  • OO Development Background
  • UML: UC Diagram, Class Diagram, Sequence Diagram
  • Analysis versus Design
• Domain Analysis and Modelling
  • Identifying Concepts (Responsibilities)
  • Attributes
  • Associations
• Domain Analysis and Modelling Exercise
Analysis versus Design

• **During Analysis**
  • we want to know about the application domain and the requirements
  • …so we develop a coarse-grained model to show where responsibilities are, and how objects interact
    • Our models show a message being passed, but we don’t worry too much about the contents of each message
    • To keep things clear, use icons to represent external objects and actors, and boxes to represent system objects

• **During Design**
  • we want to say how the software should work
  • … so we develop fine-grained models to show exactly what will happen when the system runs
    • e.g. show the precise details of each method call

What vs. How
Analysis versus Design

Use Case Models
System Sequence Diagrams
Interaction Diagrams
Domain Models (simplified Class Diagrams)
Class Diagrams

Program Code

import javax.com
import java.io.I
import java.io.I
import java.util

public class Hom
    implements Co
protected Co
protected St
public stati

public HomeA
KeyStora

}
try {
input
}
catch
LockCtrl
LightCtrl
PhotoObs

UNIVERSITY OF TARTU
INSTITUTE OF COMPUTER SCIENCE
Object-Oriented (OO) Analysis

- **OO analysis is *(claimed to be)* ‘natural’**
  - When a system evolves, the functions it performs need to be changed more often than the objects on which they operate
  - A model based on objects (rather than functions) will be more stable over time
  - Hence the claim that object-oriented designs are more maintainable
Nearly anything can be an object...

External Entities …
… that interact with the system being modeled: people, devices, other systems

Things …
… that are part of the domain being modeled: reports, displays, signals, etc.

Occurrences or Events …
… that occur in the context of the system: transfer of resources, a control action, etc. (e.g., “sale”)

Organisational Units …
… that are relevant to the application: division, group, team, etc.

Places …
… that establish the context of the problem being modeled: manufacturing floor, loading dock, etc.

Some things cannot be objects:
  procedures (e.g. print, invert, etc)
  attributes (e.g. blue, 50Mb, etc)
Object Example - ATM

Object:
ATM machine

method-1:
Accept card

method-2:
Read code

method-3:
Take selection
Object Interface

- **Interface** defines method “signatures”

- **Method signature**: name, parameters, parameter types, return type

Object *hides* its state (=attributes). The attributes are accessible only through the interface.
Cients, Servers, Messages

Objects send **messages** by calling methods

- **Client object**: sends message and asks for service
- **Server object**: provides “service” and returns result
UML Diagram Taxonomy

Structure
- Class Diagram
- Component Diagram
- Object Diagram
- Profile Diagram
- Composite Structure Diagram
- Deployment Diagram
- Package Diagram

Behavior
- Activity Diagram
- Use Case Diagram
- Interaction Diagram
- State Machine Diagram
- Sequence Diagram
- Communication Diagram
- Interaction Overview Diagram
- Timing Diagram

(Source: Wikipedia)
UML Diagram Taxonomy

(Source: Wikipedia)
Sequence Diagram

• A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline.

• Sequence diagram are good at showing which objects communicate with which other objects; and what messages trigger those communications.

• Sequence diagram are not intended for showing complex procedural logic.
Sequence Diagram

- Message - synchronous
- Message - asynchronous
- Self-message & Recursion
- Object (or Actor)
- Lifeline
- Execution Occurrence

LTAT.05.003 / Lecture 04 / © Dietmar Pfahl 2020
Use Case Diagram Example

First tier use cases

- UC1: Unlock
- UC2: Lock

Second tier use cases

- UC3: AddUser
- UC4: RemoveUser
- UC5: InspectAccessHistory
- UC6: SetDevicePrefs
- UC7: AuthenticateUser
- UC8: Login

System boundary

Actor: Tenant, Landlord

Use case: LockDevice, LightSwitch, Timer

Communication: «initiate», «participate»

«include»
# Use Case 1: Unlock

<table>
<thead>
<tr>
<th>Use Case UC-1: Unlock</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Related Requirem’ts:</strong></td>
</tr>
<tr>
<td><strong>Initiating Actor:</strong></td>
</tr>
<tr>
<td><strong>Actor’s Goal:</strong></td>
</tr>
<tr>
<td><strong>Participating Actors:</strong></td>
</tr>
</tbody>
</table>

- The set of valid keys stored in the system database is non-empty.
- The system displays the menu of available functions; at the door keypad the menu choices are “Lock” and “Unlock.”

**Postconditions:** The auto-lock timer has started countdown from autoLockInterval.

**Flow of Events for Main Success Scenario:**

1. **Tenant/Landlord** arrives at the door and selects the menu item “Unlock”
2. include::AuthenticateUser (UC-7)
3. **System** (a) signals to the **Tenant/Landlord** the lock status, e.g., “disarmed,” (b) signals to **LockDevice** to disarm the lock, and (c) signals to **LightSwitch** to turn the light on
4. **System** signals to the **Timer** to start the auto-lock timer countdown
5. **Tenant/Landlord** opens the door, enters the home [and shuts the door and locks]
System Sequence Diagram

Can be used to:

• Represent usage scenarios of actors interacting with the system-to-be (analysis)

• Represent interaction between objects within the system (design)

UC-1: unlock

Main Success Scenario

Alternate Scenario
UML Diagram Taxonomy

Structure

Class Diagram

Component Diagram

Object Diagram

Profile Diagram

Composite Structure Diagram

Deployment Diagram

Package Diagram

Notation: UML

Behavior

Activity Diagram

Use Case Diagram

Interaction Diagram

State Machine Diagram

Sequence Diagram

Communication Diagram

Interaction Overview Diagram

Timing Diagram

(Source: Wikipedia)
Classes

- A class describes a group of objects with
  - similar properties (attributes),
  - common behaviour (operations),
  - common relationships to other objects,
  - and common meaning (“semantics”).

- Example

  employee:
  - has a name, employee# and department;
  - an employee is hired, and fired; an employee works in one or more projects
Classes

- A class describes a group of objects with
  - similar properties (attributes),
  - common behaviour (operations),
  - common relationships to other objects,
  - and common meaning (“semantics”).

- Example
  employee:
  - has a name, employee# and department;
  - an employee is hired, and fired; an employee works in one or more projects
Objects vs. Classes

• The instances of a class are called objects

<table>
<thead>
<tr>
<th>Fred_Bloggs:employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: Fred Bloggs</td>
</tr>
<tr>
<td>Employee #: 234609234</td>
</tr>
<tr>
<td>Department: Marketing</td>
</tr>
</tbody>
</table>

• Two different objects may have identical attribute values (like two people with identical name and address)

• Objects have associations with other objects
  • E.g. Fred_Bloggs:employee is associated with the KillerApp:project object
  • But we will capture these relationships at the class level (why?)
Associations

- Objects do not exist in isolation from one another
  - A relationship represents a connection between things
- Types of relationships
  - Association
  - Aggregation and Composition
  - Generalization
- Class diagrams show classes and their relationships
Associations

A staff member has zero or more clients on his/her clientList. A client has exactly one staff member as a contact person.

The "liaises with" association should be read in this direction.

Role
The staff member's role in this association is as a contact person.

Role
The clients' role in this association is as a clientList.

Multiplicity
A client has exactly one staff member as a contact person.

Multiplicity
A staff member has zero or more clients on his/her clientList.

Name of the association
The name of the association is "liaises with".

Direction
The direction of the association is from staff member to client.
Multiplicity

- **Optional**  
  \(0 \text{ or } 1\) \(0..1\)

- **Exactly one**  
  \(1\) (alternative: \(1..1\))

- **Zero or more**  
  \(0..*\) (alternative: \(*\))

- **One or more**  
  \(1..*\)

- **A range of values**  
  \(1..6\)

- **A set of ranges**  
  \(1..3, 7..10, 15, 19..*\)
Association – Java Code Example

public class Person {
    private final Name name;
    private List<Address> addresses;
    //rest of Person class
}

public class Address {
    private String city;
    //rest of Address class
}

If Address objects had attributes of type Person, then this would be a bi-directional association.
Aggregation and Composition

• Aggregation (weak Association)
  • This is the “Has-a” or “Whole/part” relationship

• Composition (strong Association)
  • Strong form of aggregation that implies ownership:
    • if the whole is removed from the model, so is the part
    • the whole is responsible for the disposition of its parts
Aggregation and Composition

- **Aggregation (weak Association)**
  - This is the “Has-a” or “Whole/part” relationship.

- **Composition (strong Association)**
  - Strong form of aggregation that implies ownership:
    - if the whole is removed from the model, so is the part
    - the whole is responsible for the disposition of its parts

In IntelliJ IDEA when generating class diagrams from code you only see the black diamond!
Aggregation – Java Code Example

class Curriculum {
   private String name;
   private List<Address> addresses;

   Curriculum(String name, List<Address> addresses) {
      this.name = name;
      this.addresses = addresses;
   }

   public List<Address> getAddresses() {
      return courses;
   }
}

public String getName() {
   return name;
}

//rest of Person class

class Address {
   String city;
   String street;

   Address(String city, String street) {
      this.city = city;
      this.street = street;
   }

   public String getCity() {
      return this.city;
   }
}

//rest of Address class
Composition – Java Code Example

```java
public class Person {
    private Name name;
    ...

    public Person() {
        name = new Name();
    }

    //rest of Person class
}

public class Name {
    private String fname;
    private String lname;

    setFname(), ...

    public class Name {
        private String fname;
        private String lname;

        //rest of Room class
    }

    //rest of Person class
}
```

Name object is owned by Person object
When Person object is destroyed, Name object will no longer exist
Composition – Java Code Example

public class Person {
    private Heart heart;
    ...

    public Person() {
        heart = new Heart();
    }

    //rest of Person class
}

public class Heart {
    private double size;

    public void setSize(double size) {
        this.size = size;
    }

    //rest of Heart class
}

Heart object is owned by Person object
When Person object is destroyed, Heart object will no longer exist
Generalisation

- Sub-classes **inherit** attributes, associations, & operations from the superclass
- A sub-class may override an inherited aspect
- Super-classes may be declared **abstract**, meaning they have no instances
  - Implies that the sub-classes cover all possibilities
Generalisation – Java Code Example

```java
public class Person {
    private String name;
    private String address;

    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String getAddress() {
        return address;
    }
}

public class Employee extends Person {
    private String company;

    public void setCompany(String company) {
        this.company = company;
    }
    public String getCompany() {
        return company;
    }
}
```
Aggregation
This is the “Has-a” relationship
Composition
This is the “Whole/part” relationship
Strong form of aggregation that implies ownership:
if the whole is removed from the model, so is the part
the whole is responsible for the disposition of its parts

Can be used to model the domain structure, i.e. concepts and their properties and relationships
Structure of Lecture 04

- Preliminaries and Context
  - OO Development Background
  - UML: UC Diagram, Class Diagram, Sequence Diagram
  - Analysis versus Design
- Domain Analysis and Modelling
  - Identifying Concepts (Responsibilities)
  - Attributes
  - Associations
- Domain Analysis and Modelling Exercise
Example: ATM Machine

(a) System

Actor (Bank customer)

System

Deposit
Withdraw
Transfer

(b) Domain Model

Actor (Remote datacenter)

Actor

Concept 1

Concept 2

Concept 3

Concept n
Domain Models

• Captures the most important concepts of the domain and their associations
  • The domain is the background knowledge of the users of the system, e.g. the domain of the librarian includes books, loans, returns, patrons, …

• Helps us to understand and think about the concepts we will use in the project

• Provides a useful “glossary” for the project
Domain Models

• The domain model is also called conceptual model, domain object model or analysis object model.

• To visualize domain models the UML class diagram notation is used
  • However, no operations are defined in domain models
  • Only ...
    • domain objects and conceptual classes
    • associations between them
    • attributes of conceptual classes
Domain Model Relationships

Conceptual Class Diagram
Classes, attributes, associations

Domain Model

Use Case Model

Functional Requirements

Dynamic Behavior

Interaction Diagrams (Sequence Diagrams)

Glossary

Define terms

Domain objects

Modelling happens iteratively / start with high-level and then refine
Elements of a Domain Model

- The following elements enable us to express time invariant static business rules for a domain:

  - **Domain classes** – each domain class denotes a concept (type of object).
  
  - **Attributes** – an attribute is the description of a named slot of a specified type in a domain class; each instance of the class separately holds a value.
  
  - **Associations** – an association is a relationship between two (or more) domain classes that describes links between their object instances. Associations can have roles, describing the multiplicity and participation of a class in the relationship.
  
  - **Additional rules** – complex rules that cannot be shown with symbols can be shown with attached notes.
Simple Domain Model Example

- **Person**
  - `first name : String`
  - `last name : String`
  - `salary`
  - `+owner`
  - `1..*` association with `Company`

- **Company**
  - `name`
  - `+employer`
  - `0..1` association with `Person`

- **Role**

- **Car**
  - `type`
  - `model`
  - `name`

<<Rule>>
If a person is not employed by a company then they do not have a car.

**Domain class**
What are Domain Classes?

• Each domain class denotes a concept.
  • Concept = Descriptor for a set of things that share common properties.
• Domain Classes can be:
  • *Business objects* - things that are manipulated in the business e.g. *Order*.
  • *Real world objects* - things that the business keeps track of e.g. *Contract*, *Site*.
  • *Actors/Workers/Persons* - e.g. *Controller* and *Customer*.
  • *Events that transpire* - e.g. *Sale* and *Payment*.
• A domain class has attributes and associations with other classes.
How to identify Domain Classes?

- For the target domain, first think about class categories, then think of the classed

<table>
<thead>
<tr>
<th>Conceptual Class Category</th>
<th>Classes (for the POS system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business transactions...</td>
<td>Sale, Payment</td>
</tr>
<tr>
<td>Transaction line items...</td>
<td>SalesLineItem</td>
</tr>
<tr>
<td>Product or service related to a transaction or transaction line item.</td>
<td>Item</td>
</tr>
<tr>
<td>Where is the transaction recorded?</td>
<td>Register</td>
</tr>
<tr>
<td>Roles of people or organizations related to the transaction; actors in use cases.</td>
<td>Cashier, Customer, Store</td>
</tr>
<tr>
<td>Place of transactions.</td>
<td>Store</td>
</tr>
<tr>
<td>Noteworthy events, often with a time or place that needs to be remembered.</td>
<td>Sale, Payment</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
How to identify Domain Classes?

• An obvious way to identify domain classes is to identify nouns and phrases in textual descriptions of a domain.

Consider a use case description as follows:

1. **Customer** arrives at a **checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale**.
3. **Cashier** enters **item identifier**.
4. System records the **sales line item** and presents the **item description**, **price** and running **total**.

Class or Attribute?
Identifying Concepts (Domain Classes) from Noun Phrases (NPs)

- Vision and Scope, Glossary and Use Cases are good for this type of linguistic analysis

- However:
  - Words may be ambiguous or synonymous
  - Noun phrases may be attributes or parameters rather than classes:
    - If it stores state information or it has multiple behaviors, then it’s a class
    - If it’s just a number or a string, then it’s probably an attribute
Consider the following problem description, analyzed for Subjects, Verbs, Objects:

The ATM verifies whether the customer's card number and PIN are correct.

If it is, then the customer can check the account balance, deposit cash, and withdraw cash.

Checking the balance simply displays the account balance.

Depositing asks the customer to enter the amount, then updates the account balance.

Withdraw cash asks the customer for the amount to withdraw; if the account has enough cash,

the account balance is updated. The ATM prints the customer's account balance on a receipt.

Analyze each subject and object as follows:

- Does it represent a person performing an action? Then it's an actor, 'R'.
- Is it also a verb (such as 'deposit')? Then it may be a method, 'M'.
- Is it a simple value, such as 'color' (string) or 'money' (number)?
  Then it is probably an attribute, 'A'.
- Which NPs are unmarked? Make it 'C' for class.
- Verbs can also be classes, for example:
  Deposit is a class if it retains state information
How to identify Attributes?

A domain class sounds like an attribute if …

• It relies on an associated class for it’s identity – e.g. ‘order number’ class associated to an ‘order’ class.
  • The ‘order number’ sounds suspiciously like an attribute of ‘order’.
• It is a simple data type – e.g. ‘order number’ is a simple integer.
  • Now it really sounds like an attribute!
Domain Modelling Strategies for Concept Identification

- 'Outside-In’ Approach: First identify boundary concepts, then internal concepts.
- Internal concepts might be further classified into control and entity concepts.

- 'Setting-up-an-enterprise’ Approach: What workers need to be hired and what things acquired?
  - Start with 'worker' concepts and their responsibilities.
  - Usually (at least) one 'Controller'.
  - Distinguish between 'doing' (D) and 'knowing' (K) responsibilities.
  - Usually: D=worker and K=thing; but not always clear.
Building a Domain Model: Outside-In

Step 1: Identifying the boundary concepts

Step 2: Identifying the internal concepts
Use Cases vs. Domain Model

In use case analysis, we consider the system as a “black box”

In domain analysis, we consider the system as a “transparent box”
Use Case Diagrams

First tier use cases

- UC1: Unlock
- UC2: Lock

Second tier use cases

- UC3: AddUser
- UC4: RemoveUser
- UC5: InspectAccessHistory
- UC6: SetDevicePrefs
- UC7: AuthenticateUser
- UC8: Login

Subsystem 1: Device Control

Subsystem 2: Account Management
# Use Case 1: Unlock

<table>
<thead>
<tr>
<th>Use Case UC-1:</th>
<th>Unlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related Requirements:</td>
<td>REQ1, REQ3, REQ4, and REQ5 stated in Table 2-1</td>
</tr>
<tr>
<td>Initiating Actor:</td>
<td>Any of: Tenant, Landlord</td>
</tr>
<tr>
<td>Actor’s Goal:</td>
<td>To disarm the lock and enter, and get space lighted up automatically.</td>
</tr>
<tr>
<td>Participating Actors:</td>
<td>LockDevice, LightSwitch, Timer</td>
</tr>
</tbody>
</table>
| Preconditions: | • The set of valid keys stored in the system database is non-empty.  
• The system displays the menu of available functions; at the door keypad the menu choices are “Lock” and “Unlock.” |
| Postconditions: | The auto-lock timer has started countdown from autoLockInterval. |

## Flow of Events for Main Success Scenario:

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ 1.</td>
<td>Tenant/Landlord arrives at the door and selects the menu item “Unlock”</td>
</tr>
<tr>
<td>2.</td>
<td>include::AuthenticateUser (UC-7)</td>
</tr>
<tr>
<td>← 3.</td>
<td>System (a) signals to the Tenant/Landlord the lock status, e.g., “disarmed,” (b) signals to LockDevice to disarm the lock, and (c) signals to LightSwitch to turn the light on</td>
</tr>
<tr>
<td>← 4.</td>
<td>System signals to the Timer to start the auto-lock timer countdown</td>
</tr>
<tr>
<td>→ 5.</td>
<td>Tenant/Landlord opens the door, enters the home [and shuts the door and locks]</td>
</tr>
</tbody>
</table>

See next slide
# Example NL Requirements

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Priority</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>5</td>
<td>The system shall keep the door locked at all times, unless commanded otherwise by authorized user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed).</td>
</tr>
<tr>
<td>REQ2</td>
<td>2</td>
<td>The system shall lock the door when commanded by pressing a dedicated button.</td>
</tr>
<tr>
<td>REQ3</td>
<td>5</td>
<td>The system shall, given a valid key code, unlock the door and activate other devices.</td>
</tr>
<tr>
<td>REQ4</td>
<td>4</td>
<td>The system should allow mistakes while entering the key code. However, to resist “dictionary attacks,” the number of allowed failed attempts shall be small, say three, after which the system will block and the alarm bell shall be sounded.</td>
</tr>
<tr>
<td>REQ5</td>
<td>2</td>
<td>The system shall maintain a history log of all attempted accesses for later review.</td>
</tr>
<tr>
<td>REQ6</td>
<td>2</td>
<td>The system should allow adding new authorized persons at runtime or removing existing ones.</td>
</tr>
<tr>
<td>REQ7</td>
<td>2</td>
<td>The system shall allow configuring the preferences for device activation when the user provides a valid key code, as well as when a burglary attempt is detected.</td>
</tr>
<tr>
<td>REQ8</td>
<td>1</td>
<td>The system should allow searching the history log by specifying one or more of these parameters: the time frame, the actor role, the door location, or the event type (unlock, lock, power failure, etc.). This function shall be available over the Web by pointing a browser to a specified URL.</td>
</tr>
<tr>
<td>REQ9</td>
<td>1</td>
<td>The system should allow filing inquiries about “suspicious” accesses. This function shall be available over the Web.</td>
</tr>
</tbody>
</table>
## Extracting the Responsibilities

<table>
<thead>
<tr>
<th>Responsibility Description</th>
<th>Type</th>
<th>Concept Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinate actions of all concepts associated with a use case, a logical grouping of use</td>
<td>D</td>
<td>Controller</td>
</tr>
<tr>
<td>cases, or the entire system and delegate the work to other concepts.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Container for user’s authentication data, such as pass-code, timestamp, door identification, etc.</td>
<td>K</td>
<td>Key</td>
</tr>
<tr>
<td>Verify whether or not the key-code entered by the user is valid.</td>
<td>D</td>
<td>KeyChecker</td>
</tr>
<tr>
<td>Container for the collection of valid keys associated with doors and users.</td>
<td>K</td>
<td>KeyStorage</td>
</tr>
<tr>
<td>Operate the lock device to armed/disarmed positions.</td>
<td>D</td>
<td>LockOperator</td>
</tr>
<tr>
<td>Operate the light switch to turn the light on/off.</td>
<td>D</td>
<td>LightOperator</td>
</tr>
<tr>
<td>Operate the alarm bell to signal possible break-ins.</td>
<td>D</td>
<td>AlarmOperator</td>
</tr>
<tr>
<td>Block the input to deny more attempts if too many unsuccessful attempts.</td>
<td>D</td>
<td>Controller</td>
</tr>
<tr>
<td>Log all interactions with the system in persistent storage.</td>
<td>D</td>
<td>Logger</td>
</tr>
</tbody>
</table>

[Note: Incomplete, e.g., 'Timer' missing ...]
Domain Model (1a)

Domain concepts for subsystem #1 of safe home access

[Note: Incomplete, e.g., Logger, Timer missing ...]
Domain Model (1b)
## Extracting Associations

<table>
<thead>
<tr>
<th>Concept pair</th>
<th>Association description</th>
<th>Association name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller ↔ StatusDisplay</td>
<td>Controller sends current status of locks etc. to StatusDisplay for display</td>
<td>...</td>
</tr>
<tr>
<td>Controller ↔ KeycodeEntry</td>
<td>KeycodeEntry device reads Input and sends it to Controller for further processing</td>
<td>...</td>
</tr>
<tr>
<td>Key ↔ Controller</td>
<td>Controller obtains Key based on Input and metadata (door, time)</td>
<td>obtains</td>
</tr>
<tr>
<td>KeyChecker ↔ Controller</td>
<td>Controller sends Key for checking to KeyChecker and receives check result for further action</td>
<td>conveys requests</td>
</tr>
<tr>
<td>KeyStorage ↔ KeyChecker</td>
<td>KeyChecker retrieves valid keys from KeyStorage for comparison with Key</td>
<td>retrieves valid keys</td>
</tr>
<tr>
<td>Key ↔ KeyChecker</td>
<td>KeyChecker verifies Key</td>
<td>verifies</td>
</tr>
<tr>
<td>HouseholdDeviceOperator ↔ Controller</td>
<td>Controller activates Devices through requests to HouseholdDeviceOperator</td>
<td>conveys requests</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
# Extracting Attributes

<table>
<thead>
<tr>
<th>Concept</th>
<th>Attributes</th>
<th>Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller</td>
<td>numOfAttempts</td>
<td>Used to determine whether another attempt is allowed</td>
</tr>
<tr>
<td></td>
<td>maxNumOfAttempts</td>
<td>Used to limit the retry attempts</td>
</tr>
<tr>
<td>HouseholdDeviceOperator</td>
<td>deviceStatuses</td>
<td>On, off, etc.</td>
</tr>
<tr>
<td>Key</td>
<td>userIdentityCode</td>
<td>Personal code of the resident</td>
</tr>
<tr>
<td></td>
<td>timestamp</td>
<td>Allows for tracking history</td>
</tr>
<tr>
<td></td>
<td>doorLocation</td>
<td>Allows possibility of different codes for doors and also relevant for tracking history</td>
</tr>
<tr>
<td>KeyChecker</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Domain Model (2a)

Domain model for UC-1: Unlock

**Associations:** who needs to work together, *not how* they work together

Concept pair | Association description | Association name

---

**Domain Model Diagram:**

- **KeyChecker**
  - **KeyStorage**
    - **Key**
      - **Attributes**
        - `userIdentityCode`
        - `timestamp`
        - `doorLocation`

- **Controller**
  - **HouseholdDeviceOperator**
    - **deviceStatuses**
      - values, e.g.: activated, stopped

- **KeycodeEntry**
- **StatusDisplay**

- **LockDevice**
- **LightSwitch**

---

“Reading direction arrow.”

Has no meaning; it only helps reading the association label, and is often left out.
Summary: How to create a Domain Model?

Perform the following in very short iterations:

- Make a list of candidate domain classes.
- Draw these classes in a UML class diagram.
- If possible, add brief descriptions for the classes.
- Identify any associations that are necessary.
- Decide whether some domain classes are really just attributes.
- Where helpful, identify role names and multiplicity for associations.
- Add any additional static rules as UML notes that cannot be conveyed with UML symbols.
- Group diagrams/domain classes by category into packages.
- Concentrate more on just identifying domain classes in early iterations!
General Recommendations

• Identifying the domain concepts is more important than identifying their associations and attributes
  • Every concept that the designer can discover, should be mentioned.
• For an association (or attribute), in order to be shown it should pass the “does it need to be mentioned?” test.
  • If the association in question is obvious, it should be omitted from the domain model.
  • For example, the association <<Controller–obtains–Key>> is fairly redundant.
  • Several other associations could as well be omitted, because the reader can easily infer them, and this should be done particularly in schematics that are about to become cluttered.
• Clarity should be preferred over detail!
More examples with explanation can be found in the Textbook
Structure of Lecture 04

• Preliminaries and Context
  • OO Development Background
  • UML: UC Diagram, Class Diagram, Sequence Diagram
  • Analysis versus Design
• Domain Analysis and Modelling
  • Identifying Concepts (Responsibilities)
  • Attributes
  • Associations
• Domain Analysis and Modelling Exercise
Statements about a Course Management System

• During a semester a lecturer reads one or more lectures
• Sometimes the lecturer is on leave to focus on doing research, in this case (s)he does not give a lecture
• A student usually attends one or more lectures, unless (s)he has something better to do
• During the semester there will be several exercises which are meant to be solved by small study groups
• Each student is assigned to one particular study group for the whole semester
• A study group consists of two to three students
• After submission of a solution by a study group it is graded by a tutor
• …
A class describes a set of objects with the same semantics, properties and behavior.

When used for domain modeling, it is a visualization of a real world concept.

- During a semester a lecturer reads one or more lectures.
- A student usually attends one or more lectures, ...
- During the semester there will be several exercises...
- Each student is assigned to one particular study group for the whole semester.
- ... it is graded by a tutor.
A class describes a set of objects with the same semantics, properties and behavior

When used for domain modeling, it is a visualization of a real world concept

- During a **semester** a **lecturer** reads one or more **lectures**
- A **student** usually attends one or more lectures, ...
- During the semester there will be several **exercises**...
- Each student is assigned to one particular **study group** for the whole semester
- ... it is graded by a **tutor**
Attributes are logical data values of an object
Attributes are logical data values of an object

- Semester
- Lecturer name
- Lecture description
- Student name
- Exercise tasks submit dates points

(incomplete)
An association is a relationship between classes.

The ends of an association are called roles. Roles optionally have a multiplicity, name and navigability.

- During a semester a lecturer reads one or more lectures
- A student usually attends one or more lectures, unless (s)he has something better to do
- Each student is assigned to one particular study group for the whole semester
- A study group consists of two to three students
An association is a relationship between classes.

The ends of an association are called roles. Roles optionally have a multiplicity, name and navigability.

- During a semester a **lecturer** reads one or more lectures
- A **student** usually **attends** one or more **lectures**, unless (s)he has something better to do
- Each **student** is **assigned to** one particular **study group** for the whole semester
- A **study group** **consists of** two to three **students**

(incomplete)
First Draft of Domain Model

(links and associations are incomplete)
More examples of domain modeling

POS

- http://csis.pace.edu/~marchese/CS616/Lec5/se_l5a.htm

Course management and POS


Bank accounts

- http://documentation.genesez.org/javaee/de.genesez.uml.modeling.domain.html
Next Lecture

• Date/Time:
  • Friday, 02-Oct, 10:15-12:00

• Topic:
  • Development Infrastructure (by Stepan Bolotnikov, Proekspert AS)

• For you to do:
  • Work on homework 2 assignment
  • Remember that all team members must be present in next week’s lab sessions (assessment of homework 2 progress)!
  • Watch last year’s lecture video and write questions (if you have any) to the general channel in Slack so that Kristiina can pick them up for her Q&A session during lecture time next week